Pre-copy-edited version of paper that appeared in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 11. Nov. 1998, pp. 1088-1101.

A Stochastic Model for Heterogeneous Computing and Its Application in Data Relocation Scheme Development

Min Tan [*] and *Howard Jay Siegel* [†]

[*] Segue Software Inc.
142 South Santa Cruz Ave.
Los Gatos, CA 95030
mtan@segue.com

[†] Parallel Processing Laboratory
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285, USA
hj@purdue.edu

Submitted in March 1997
Revised in June 1998

Abstract

In a dedicated mixed-machine heterogeneous computing (HC) system, an application program may be decomposed into subtasks, then each subtask assigned to the machine where it is best suited for execution. Data relocation is deTned as selecting the sources for needed data items. It is assumed that multiple independent subtasks of an application program can be executed concurrently on different machines whenever possible. A theoretical stochastic model for HC is proposed, in which the computation times of subtasks and communication times for inter-machine data transfers can be random variables. The optimization problem for Tnding the optimal matching, scheduling, and data relocation schemes to minimize the total execution time of an application program is deTned based on this stochastic HC model. The global optimization criterion and search space for the above optimization problem are described. It is validated that a greedy algorithm based approach can establish a local optimization criterion for developing data relocation heuristics. The validation is provided by a theoretical proof based on a set of common assumptions about the underlying HC system and application program. The local optimization criterion established by the greedy approach, coupled with the search space deTned for choosing valid data relocation schemes, can help developers of future practical data relocation heuristics.

Keywords: data relocation, greedy algorithm, heterogeneous computing, mapping, matching, optimization, scheduling, stochastic modeling.

This research was supported in part by the DARPA/ITO Quorum Program under NPS subcontract number N62271-98-M-0217, and by NRaD

under contract number N66001-96-M-2277.

1: Introduction

A single application program often requires many different types of computation that result in different needs for machine capabilities. Heterogeneous computing (HC) is the effective use of the diverse hardware and software components in a heterogeneous suite of machines connected by a high-speed network to meet the varied computational requirements of a given application [CiL95, FrS93, KhP93, SiA96, SiD97, ZhY95]. One goal of HC is to decompose an application program into subtasks, each of which is computationally homogeneous, and then assign each subtask to the machine where it is best suited for execution.

Subtask matching, scheduling, and data relocation are three critical steps for implementing an HC application on an HC system. Matching involves assigning subtasks to machines. Scheduling includes ordering the execution of the subtasks assigned to each machine and ordering the inter-machine communication steps for data transfers. Data relocation is the scheme for selecting the sources for needed data items. Here, a stochastic HC model is developed and used as a basis to study theoretical issues for data relocation. The practical implication of the theoretical results derived on data relocation heuristic design is explained. It is assumed that multiple independent subtasks of an application program can be executed concurrently on different machines whenever possible (e.g., when the machines are available for subtask execution).

The contribution of this paper can be summarized as follows. A theoretical stochastic HC model is proposed, in which the computation times of subtasks and communication times for inter-machine data transfers are modeled as random variables. The rest of this paper focuses on theoretical issues for data relocation using a stochastic HC model. The optimization problem for finding the optimal matching, scheduling, and data relocation schemes to minimize the total execution time of an application program executed in a dedicated HC system is defined based on this proposed stochastic HC model. The global optimization criterion and search space for the above optimization problem in HC are described. It is validated that a greedy algorithm based approach can establish a local optimization criterion for developing data relocation heuristics in

practice. The validation is provided by a theoretical proof based on a set of common assumptions about the underlying HC system and application program. The local optimization criterion established by the greedy approach, coupled with the search space deTned for choosing valid data relocation schemes, can help developers of future practical data relocation heuristics.

The inter-machine communication time between subtasks can be substantial and is one of the major factors that degrade the performance of an HC system. This paper focuses on potential methods for minimizing the inter-machine communication time of an application program when the concurrent execution of different subtasks on different machines is considered whenever possible. In particular, the impact of the data relocation scheme on the total execution time of the subtasks executed in a dedicated HC system is examined.

In most of the mathematical models for HC in the literature (e.g., [ChE93, IvO95, NaY94, TaA95, TaS97, WaK92]), the computation times and inter-machine data transfer times of data items for different subtasks in the application program are assumed to be deterministic quantities. This is valid when the inter-machine network is completely controlled by the scheduler and all execution times and inter-machine communication needs are known *a priori* (not dependent on input data). However, there are elements of uncertainty (e.g., input-data-dependent conditional and looping constructs) that impact the deterministic nature of both the computation and inter-machine communication times for different subtasks. Such uncertainties can create others, e.g., network contention among different inter-machine data transfer steps. They are unpredictable prior to execution time. One approach to modeling these computation and communication times is to represent them as random variables with assumed probability distribution functions.

To use a dedicated HC system to execute an application program efTciently, the optimization problem of using matching, scheduling, and data relocation schemes to minimize the total execution time must be deTned. Section 2 provides the background and terminology needed for the rest of this paper. In Section 3, a theoretical stochastic HC model for matching, scheduling, and data relocation is introduced. Based on the random variables of the HC model and given matching, scheduling, and date relocation schemes, a procedure for determining the execution

time of an application program (with partially ordered subtasks) is presented in Section 4. In Section 5, a method is devised to enumerate all the valid options in choosing the data relocation scheme for a given arbitrary matching. The cases in which the application programs may include inter-subtask conditional and inter-subtask looping constructs are considered. Thus, Sections 3, 4, and 5 collectively deTne the above optimization problem in HC with a stochastic model. Because of the complexity of this deTned optimization problem in HC, guidelines for devising heuristics must be provided. It is validated in Section 6 that a greedy algorithm based approach can establish a local optimization criterion for developing data relocation heuristics. The validation is provided by a theoretical proof based on a set of common assumptions about the underlying HC system and application program. This theoretical result indicates that a greedy algorithm based approach can achieve reasonable local optimization for developing data relocation heuristics in practice.

Most of the literature for HC has concentrated on addressing the practical aspects and heuristics for matching and scheduling. This paper emphasizes instead the theoretical issues involved in data relocation using a stochastic HC model. The practical implication of the theoretical results derived on data relocation heuristic design is explained.

This research was supported in part by the DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks). MSHN is a collaborative research effort among NPS (Naval Postgraduate School), Noemix (a company specializing in software technology for distributed computing), Purdue University, and USC (University of Southern California). It builds on SmartNet [FrG98], an operational scheduling framework and system for managing resources in a heterogeneous environment developed at the NRaD naval laboratory, which also supported this research. The technical objective of MSHN project is to design, prototype, and reTne a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service.

2: Background and Terminology

The material in this subsection is summarized from [TaS97]. It provides the background and terminology needed for the rest of this paper. In general, the goal for HC is to assign each subtask to one of the machines in the system such that the total execution time (computation time and inter-machine communication time) of the application program is minimized [ChE93, Fre89, NaY94, WaK92]. The subtask to machine assignment problem is referred to as matching in HC. When a subset of subtasks can be executed in any order, varying the order of the computation of these subtasks (while maintaining the data dependencies among all subtasks) can impact the total execution time of the application program. Determining the order of computation for the subtasks is referred to as scheduling in HC. In most of the literature for HC, a subtask ţow graph is used to describe the data dependencies among subtasks in an application program (e.g., [IvO95, NaY94, SiL93, TaA95, TaS97, Tow86]). In Figure 1, each vertex of the subtask ţow graph represents a subtask. Let $S[k]$ denote the k-th subtask. For each data element that $S[k]$ transfers to $S[j]$ during execution, there is an edge from $S[k]$ to $S[j]$ labeled with the corresponding variable name. An extra vertex labeled *Source* denotes the locations where the initial data elements of the program are stored.

Let a data item be a block of information that can be transferred between subtasks. Using information from the subtask ţow graph, a data item is denoted by the two-tuple (s, d), where $s \geq 0$ is the number of the subtask that generates the needed value of variable d upon completion of computation of that subtask. If the needed value of d is an initial data element to the program, then $s = -1$. Two data items are the same if and only if they are both associated with the same variable name in an application program and the corresponding value of the data is generated by the same subtask (which implies that the two data items have the same value).

In general, most of the graph-based algorithms for matching-related problems assume that the pattern of data transfers among subtasks is known *a priori* and can be illustrated using a subtask ţow graph (e.g., [IvO95, NaY94, SiL93, Tow86]). Thus, no matter which machine is used

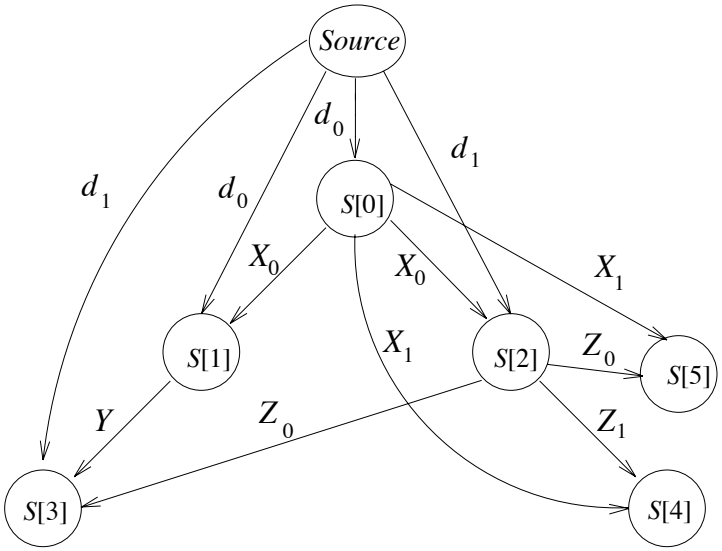

Figure 1: Subtask ţow graph for the example application program.

for executing each subtask of a speciŢc application program, the locations (subtasks) from which each subtask obtains its corresponding input data items are determined by the subtask ţow graph and are independent of any particular matching scheme between machines and subtasks.

The above assumption generally needs reŢnement in the case of HC. In [TaS97], two data-distribution situations, namely data locality and multiple data-copies, are identiŢed for addressing reŢnements of the above assumption. It is assumed that each subtask $S[i]$ keeps a copy of each of its individual input data items and output data items on the machine to which $S[i]$ is assigned by the matching scheme. Furthermore, it is also assumed that all input data items are received for a subtask prior to that subtask's computation.

Data locality arises when two subtasks, $S[j]$ and $S[k]$ that are assigned to the same machine, need the same data item e from $S[i]$ (assigned to a different machine). Because a machine can fetch a data item from its local storage faster than fetching it from other machines, if $S[j]$ is executed after $S[k]$, then $S[j]$ should obtain e locally from $S[k]$ instead of from the machine assigned to $S[i]$. If a subtask ţow graph is used to compute inter-subtask communication cost, then

without considering machine assignments, the impact of data locality might be ignored.

The multiple data−copies situation arises when two subtasks, $S[j]$ and $S[k]$, need the same data item e from $S[i]$, where $S[i]$, $S[j]$, and $S[k]$ are assigned to three different machines. If $S[k]$ is executed after $S[j]$ obtains e, then the machine assigned to $S[k]$ can get data item e from either the machine assigned to $S[i]$ or the machine assigned to $S[j]$. The choice that results in the shorter time should be selected. Selecting the sources for needed data items is referred to as data relocation (because the data relocation scheme determines the source machines from which the data items will be relocated to the destination machines). In general, when using information only from the subtask ţow graph, the possibility of having multiple sources for a needed data item is not considered. Data locality can be viewed as a special case of having multiple data copies (i.e., one copy is on the machine to which the receiving subtask is assigned by the matching scheme).

In [TaS97], it is assumed that, at any instant in time during the execution of an application program, only one inter-machine data transfer step is being executed. All computation and inter-machine communication times of subtasks are assumed to be known deterministic quantities, and any data conditional and looping constructs must be contained within a single subtask. Based on these assumptions, a minimum spanning tree based algorithm is presented in [TaS97] that Ţnds, for a given matching, the optimal scheduling scheme for inter-machine data transfer steps and the optimal data relocation scheme for each subtask. Data locality and multiple data-copies are all considered in the above algorithm. The mathematical model for HC presented in this paper differs from the one in [TaS97] in that, here, limited only by inter-subtask data-dependencies and machine assignments, at any instant in time, multiple subtasks can be executed and multiple inter-machine data transfers can be performed. Also, here the computation times of subtasks and communication times for inter-machine data transfers can be random variables. Furthermore, the cases in which the application programs may include inter-subtask conditional and inter-subtask looping constructs are considered. Thus, the HC model presented here is much more general than the one in [TaS97], which makes the data relocation more complex. It is vali-

dated in this paper that a greedy algorithm based approach can establish a local optimization criterion for developing data relocation heuristics. This result indicates that a greedy algorithm based approach can achieve reasonable local optimization for developing data relocation heuristics in practice.

3: A Stochastic Model for Matching, Scheduling, and Data Relocation in HC

A stochastic model of matching, scheduling, and data relocation for HC is formalized in this section. This model is an extension of the one presented in [TaS97]. The possible concurrent execution of both the computation of subtasks and inter-machine communication steps in an application program is considered. The issues related to using a theoretical stochastic HC model are addressed. When the computation time of each subtask on each machine and the communication times of transferring data items have stochastic properties, those timing parameters must be modeled as random variables. This paper examines underlying theoretical issues with respect to data relocation. Due to the theoretical nature of the proof of the main result in this paper, it is not necessary to know the actual distribution functions of those random variables. The mathematical model presented in this section allows the material in the rest of this paper to be given in unambiguous terms. All notation developed in the remaining sections is summarized in the appendix for the glossary of notation at the end of this paper.

(1) An application program $\underline{P}$ is composed of a set of $\underline{n}$ subtasks

$$\underline{S} = \{S[0], S[1], ..., S[n-1]\}.$$

There are a set of $\underline{Q}$ initial data elements

$$\{d_0, d_1, ..., d_{Q-1}\}.$$

(2) Suppose that $\underline{NI[i]}$ is the number of input data items required by $S[i]$ and $\underline{NG[i]}$ is the number of output data items generated by $S[i]$. There are two sets of data items associated with each $S[i]$. One is the input data set

$$\underline{I[i]} = \{Id[i, 0], Id[i, 1], ..., Id[i, NI[i] - 1]\},$$

the other is the generated output data set

$$\underline{G[i]} = \{Gd[i, 0], Gd[i, 1], ..., Gd[i, NG[i] - 1]\}.$$

The program structure of P is speciTed by a subtask tow graph.

In this paper, the subtask tow graph of any application program P is assumed to be acyclic. A cycle in a graph represents a loop containing one or more subtasks. With the presence of the inter-subtask looping constructs, an appropriate statistical approach can be used to determine the distribution for the number of iterations each looping construct will execute and the maximum number of iterations each looping construct has [Tow86]. Then, the existent subtask tow graph can be transformed into an acyclic one by unrolling each looping construct with the known or estimated maximum number of iterations. This is the approach presented in Subsection 5.3.2. The above approach potentially will increase the number of subtasks present in the acyclic subtask tow graph signiTcantly. Also, the distribution for the number of iterations each looping construct will execute and the maximum number of iterations each looping construct has can be difTcult to estimate in reality. A possibly more practical approach is to group a Txed number of consecutive iterations of each unrolled looping construct together as a single subtask to decrease the number of subtasks present. Another approach is to view each looping construct as part of a single subtask and the boundaries for decomposing an application program into subtasks are not allowed to be in the middle of a looping construct.

(3) An HC system consists of a heterogeneous suite of $\underline{m}$ machines

$$\underline{M} = \{M[0], M[1], ..., M[m - 1]\}.$$

M includes the devices where all the initial data elements are stored before the execution of the application program P.

(4) There is a computation matrix $\underline{C} = \{C[i, j]\}$, where $\underline{C[i, j]}$ denotes the computation time of $S[i]$ on machine $M[j]$ (e.g. [GhY93, YaK94]). For the reason stated in Section 1, $C[i, j]$ is

assumed to be a random variable with a known distribution. It can be computed from empirical information or by applying two characterization techniques in HC, namely task proTling and analytical benchmarking (see [SiA96] for a survey of these techniques). In [LiA95], a methodology is introduced for estimating the distribution of execution time for a given data parallel program that is to be executed on a single hybrid SIMD/SPMD mixed-mode machine. This methodology is extended in [LiA97] for estimating the distribution of execution time for an application program that is to be executed on a mixed-machine HC system. However, as mentioned earlier, for the results mentioned here, it is not necessary to determine the distribution functions for the random variables.

(5) The matching associated with the application program P is deTned by an assignment function $\underline{Af}$: $S \rightarrow M$ such that if $Af(i) = j$, then $S[i]$ is assigned to be executed on machine $M[j]$. $\underline{NS[j]}$ is deTned as the number of subtasks assigned to be executed on machine $M[j]$. Thus, $\sum_{j=0}^{m-1} NS[j] = n$.

(6) A scheduling function $\underline{Sf}$ indicates the execution order of a subtask with respect to the other subtasks assigned to the same machine. If $Sf(i) = k$, then $S[i]$ is the k-th subtask whose computation is executed on machine $M[Af(i)]$, where $0 \leq k < NS[Af(i)]$. Readers should notice that the scheduling function Sf schedules only the order of the computation for different subtasks (*not* the order for executing the inter-machine communication steps).

(7) The set of data−source functions is

$$\underline{DS} = \{DS[0], DS[1], ..., DS[n-1]\},$$

where $DS[i](j) = [k_1, k_2]$ ($0 \leq i < n$, $0 \leq j < NI[i]$, $0 \leq k_1 < n$, and $0 \leq k_2 < m$) means that $S[i]$ obtains the input data item $Id[i, j]$ from $S[k_1]$ and $k_2 = Af(k_1)$. If $DS[i](j) = [k_1, k_2]$ and $k_1 = -1$, then $Id[i, j] = (-1, d_x)$ and $S[i]$ obtains the associated initial data element from machine $M[k_2]$ where d_x is initially stored. Readers should notice that, when $k_1 \neq -1$, the augmented information k_2 can be obtained with the known Af and is redundant. But the in-

formation from k_2 is necessary to specify the source of an initial data element when $k_1 = -1$. The above definition of *DS* gives a unified way of specifying the values of a data-source function. If each subtask fetches its input data items only from the sources where they are generated (in the case of the initial data elements, from their initial locations), there exists only one choice of *DS* for each specific *Af* and *Sf*. But if the impact of the data locality and multiple data-copies is considered, there are different choices for *DS*. This choice of *DS* corresponds to the data relocation problem discussed in Section 2.

It is assumed that each subtask $S[i]$ will submit a copy of its input data item $Id[i, j]$ to the network for forwarding to other destination machines (based on *DS*) immediately after $Id[i, j]$ is available on machine $M[Af(i)]$. Each subtask will also submit copies of all of its output data items to the network to be transferred to the proper destination machines (based on *DS*) after the completion of its entire computation. Thus, *Af*, *Sf*, and *DS* together completely specify the computation and inter-machine communication steps needed at any time to execute the application program *P* in a dedicated HC system.

(8) The communication time estimator $D[s, r, e]$ denotes the length of the communication time interval between the time when a data item e is available on $M[s]$ and the time when e is obtained by $M[r]$ (assuming this transfer is required for the given *Af*, *Sf*, and *DS*). For the reason stated in Section 1, $D[s, r, e]$ is assumed to be a random variable (again recall that the distribution of this random variable is not needed to derive the results of this paper). $D[s, r, e]$ includes all the various hardware and software related times of the inter-machine communication process (e.g., network latency and the time for data format conversion between $M[s]$ and $M[r]$ when necessary).

Most of the literature for HC (e.g., [GhY93, KhP92, TaA95, TaS97]) assumes that the inter-machine communication time for sending a data item e from $M[s]$ to $M[r]$ is only a function of s, r, and e. But in reality, even in a dedicated HC system, when an application program is executed, the traffic pattern for inter-machine communication can be impacted by subtask com-

putation and other inter-machine communication times that are all input data dependent (and represented as random variables). The choice of *Af*, *Sf*, and *DS* impacts all of these computation and communication times and, hence, the communication time interval between the time when *e* is available on *M*[*s*] and the time when *e* is obtained by *M*[*r*]. Thus, the communication time estimator *D*[*s*, *r*, *e*] is dependent on *Af*, *Sf*, *DS*, *s*, *r*, and *e*.

In general, it will be extremely difficult (if not impossible) to estimate the distribution function of *D*[*s*, *r*, *e*] as a function of *Af*, *Sf*, *DS*, *s*, *r*, and *e*. The purpose of defining *D*[*s*, *r*, *e*] here is to address the factors that impact the inter-machine communication times for the application programs executed in a dedicated HC system. It also helps to establish a theoretical model for defining the global optimization criterion of the optimization problem for HC. With this well-defined theoretical model and global optimization criterion, the greedy algorithm based approach introduced in Section 6 can provide potential data relocation heuristics with a sound local optimization criterion based on a solid theoretical derivation. Within the matching and scheduling problem domain, many researchers have shown that local optimization is a worthwhile approach to achieve global optimization (e.g., [ElL90, IvO95, SiL93]). Thus, future data relocation heuristics can follow the local optimization criterion in Section 6 to achieve a reasonable level of global optimization without the information about the exact distribution function of *D*[*s*, *r*, *e*].

4: A Topological Sort Based Algorithm for Calculating the Execution Time of an Application Program in an HC System

In this section, a topological sort based algorithm for calculating the total execution time (computation and communication times) of an HC application program is introduced. This algorithm helps establish the global optimization criterion of the optimization problem for HC with respect to matching, scheduling, and data relocation.

For a given computation matrix *C* and communication time estimator *D*[*s*, *r*, *e*], the total execution time of the application program *P* associated with an assignment function *Af*, a

scheduling function *Sf*, and a set of data-source functions *DS* is deṬned by the following procedure. A data relocation graph (denoted as *Gr*) corresponding to a particular *Af*, *Sf*, and *DS* is generated using the steps speciṬed below. When the impact of data locality and multiple data-copies is considered, the concept of a valid set of data-source functions *DS* of the application program *P* can be deṬned according to the properties of *Gr*. There may be many valid sets for *P*, each corresponding to a unique graph for *P*, and each resulting in possibly different execution time of *P*. An invalid *DS* would correspond to a set of data-source functions that does not result in an operational program.

The steps for constructing *Gr* are as follows.

Step 1: A *Source* vertex is generated that represents the locations of all the initial data elements (which may be on different machines).

Step 2: For each $S[i]$, $NI[i] + 1$ vertices are created, one for each of the $NI[i]$ input data items and one for all of the generated output data items of $S[i]$. These are the set of input data vertices, labeled $V[i, j]$ $(0 \le j < NI[i])$ and the output data vertex $V_g[i]$ (as shown in Figure 2). $V[i, j]$ represents the operation for subtask $S[i]$ to receive its j-th input data item. $V_g[i]$ represents the computation for $S[i]$ to generate all of its output data items. V is a set that contains all of the above vertices associated with the application program *P* in Steps 1 and 2. Each $V[i, j]$ is associated with a weight zero and each $V_g[i]$ is associated with a weight $C[i, Af(i)]$, the computation time of subtask $S[i]$ on the machine assigned by the assignment function *Af*.

Step 3: For any input data vertex $V[i_1, j_1]$, suppose that $DS[i_1](j_1) = [i_2, k_2]$ where $-1 \le i_2 < n$ and $0 \le k_2 < m$, and if $0 \le i_2 < n$, then $k_2 = Af(i_2)$.

Case A: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from the *Source* vertex if $Id[i_1, j_1] = (-1, d_k)$ and d_k is one of the initial data elements.

If $i_2 = -1$, then there exists k $(0 \le k < Q)$, such that $Id[i_1, j_1] = (-1, d_k)$, and a directed edge with weight $D[k_2, Af(i_1), Id[i_1, j_1]]$ is added from the *Source* vertex to $V[i_1, j_1]$ (recall that $DS[i_1](j_1) = [i_2, k_2]$ implies that d_k is received from machine $M[k_2]$). That is, if subtask $S[i_1]$'s j_1-th input data item $Id[i_1, j_1]$ is one of the initial data elements and is obtained from

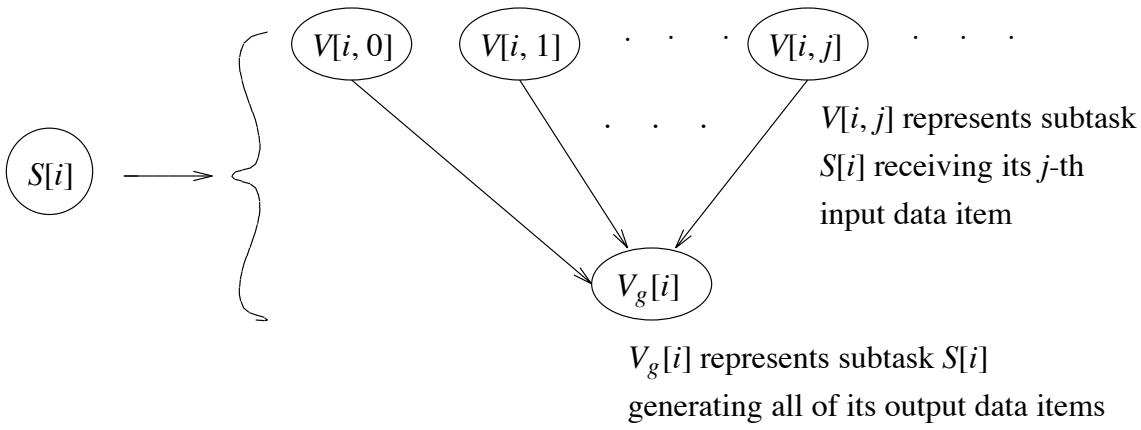

Figure 2: The generation of the input and output data vertices and activate edges for $S[i]$.

one of the initial locations where d_k is stored before program execution, then add an edge from the *Source* vertex to $V[i_1, j_1]$ whose weight is the communication time interval needed to transfer that initial data element from the initial location $M[k_2]$ where it is stored to the machine assigned to $S[i_1]$.

Case B: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from the subtask that generates $Id[i_1, j_1]$.

If $0 \le i_2 < n$ and there is j_2, such that $Id[i_1, j_1] = Gd[i_2, j_2]$, then a directed edge with weight $D[k_2, Af(i_1), Id[i_1, j_1]]$ is added from $V_g[i_2]$ to $V[i_1, j_1]$. That is, if subtask $S[i_1]$'s j_1-th input data item $Id[i_1, j_1]$ is subtask $S[i_2]$'s j_2-th output data item $Gd[i_2, j_2]$, then add an edge from $V_g[i_2]$ to $V[i_1, j_1]$ whose weight is the communication time interval needed to transfer that data item from $M[k_2]$ to the machine assigned to $S[i_1]$.

Case C: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from one of the other subtasks that have obtained that input-data item already.

If $0 \le i_2 < n$, and there is a j_2, such that $Id[i_1, j_1] = Id[i_2, j_2]$, then a directed edge with weight $D[k_2, Af(i_1), Id[i_1, j_1]]$ is added from $V[i_2, j_2]$ to $V[i_1, j_1]$. That is, if subtask $S[i_1]$'s j_1-th input data item $Id[i_1, j_1]$ is obtained by copying subtask $S[i_2]$'s j_2-th input data item $Id[i_2, j_2]$,

then add an edge from $V[i_2, j_2]$ to $V[i_1, j_1]$ whose weight is the communication time interval needed to transfer that data item from $M[k_2]$ to the machine assigned to $S[i_1]$.

For any input data vertex $V[i_1, j_1]$ ($0 \le i_1 < n$ and $0 \le j_1 < NI[i_1]$) for a given *DS*, one and only one case of A, B, or C can occur. Thus, any vertex $V[i_1, j_1]$ has one and only one parent vertex, which is specified by the given *DS*. Also, the weight of the edge between $V[i_1, j_1]$ and its unique parent vertex is the communication time interval needed for $S[i_1]$ to obtain $Id[i_1, j_1]$ from its source with respect to the given *Af*, *Sf*, and *DS*.

Step 4: For every $0 \le i < n$, a directed edge with weight zero is added from $V[i, j]$ to $V_g[i]$ for all j, $0 \le j < NI[i]$ (as shown in Figure 2). All the edges generated in this step are called activate edges.

As an example, suppose that for the specific application program *P* illustrated by the subtask flow graph shown in Figure 1, Table 1 lists its corresponding parameters. The initial data elements of *P* are d_0 and d_1; The generated data items of *P* are X_0, X_1, Y, Z_0, and Z_1. Note that initial data elements are named with lower case letters and generated data items with upper case letters. The result of applying the set of data-source functions defined by the subtask flow graph in Figure 1 is shown by Figure 3 (recall that is just one possible set of data-source functions).

If the *Gr* generated above is an acyclic graph, then the corresponding *DS* is defined as a valid set of data−source functions for the application program *P*. If the graph had a cycle, then deadlock would arise in the application program *P*, which makes *P* unschedulable. Readers should notice that the weight of each edge or vertex depends on *Af*, *Sf*, and *DS*. The validity of a particular *DS* is based on the subtask flow graph and is independent of the underlying *Af* and *Sf* for generating the specific *Gr*. For the rest of this paper, only valid sets of data-source functions will be considered.

Step 5: For each i_1 and i_2 ($0 \le i_1 < n$ and $0 \le i_2 < n$), if $Af(i_1) = Af(i_2)$ and $Sf(i_1) = Sf(i_2) - 1$ (i.e., $S[i_1]$ and $S[i_2]$ are assigned to the same machine and $S[i_1]$ is executed immediately before

subtask	no. inputs	input data items	no. outputs	output data items
S [0]	$NI[0] = 1$	Id [0, 0] = $(-1, d_0)$	$NG[0] = 2$	Gd [0, 0] = $(0, X_0)$ Gd [0, 1] = $(0, X_1)$
S [1]	$NI[1] = 2$	Id [1, 0] = $(-1, d_0)$ Id [1, 1] = $(0, X_0)$	$NG[1] = 1$	Gd [1, 0] = $(1, Y)$
S [2]	$NI[2] = 2$	Id [2, 0] = $(0, X_0)$ Id [2, 1] = $(-1, d_1)$	$NG[2] = 2$	Gd [2, 0] = $(2, Z_0)$ Gd [2, 1] = $(2, Z_1)$
S [3]	$NI[3] = 3$	Id [3, 0] = $(-1, d_1)$ Id [3, 1] = $(1, Y)$ Id [3, 2] = $(2, Z_0)$	$NG[3] = 0$	
S [4]	$NI[4] = 2$	Id [4, 0] = $(0, X_1)$ Id [4, 1] = $(2, Z_1)$	$NG[4] = 0$	
S [5]	$NI[5] = 2$	Id [5, 0] = $(0, X_1)$ Id [5, 1] = $(2, Z_0)$	$NG[5] = 0$	

Table 1: Parameters for the subtask ţow graph shown in Figure 1.

$S[i_2]$), a directed edge with weight zero is added from $V_g[i_1]$ to $V_g[i_2]$. The extended graph based on *Gr* and *Sf* after this step is deŢned as the execution graph *Ex* of *P*. For the example in Figure 1, one possible assignment function *Af* is: *Af*(0) = 1, *Af*(1) = 2, *Af*(2) = 2, *Af*(3) = 1, *Af*(4) = 3, and *Af*(5) = 0. One possible scheduling function *Sf* for this example and *Af* is: *Sf*(0) = 0, *Sf*(1) = 0, *Sf*(2) = 1, *Sf*(3) = 1, *Sf*(4) = 0, *Sf*(5) = 0, then the corresponding directed edges added by this step are shown by the dashed lines from $V_g[0]$ to $V_g[3]$ and from $V_g[1]$ to $V_g[2]$ in Figure 3. If the generated execution graph *Ex* is acyclic, then the corresponding scheduling function generates an operational program and is deŢned as a valid scheduling function. For the rest of this paper, only valid scheduling functions will be considered.

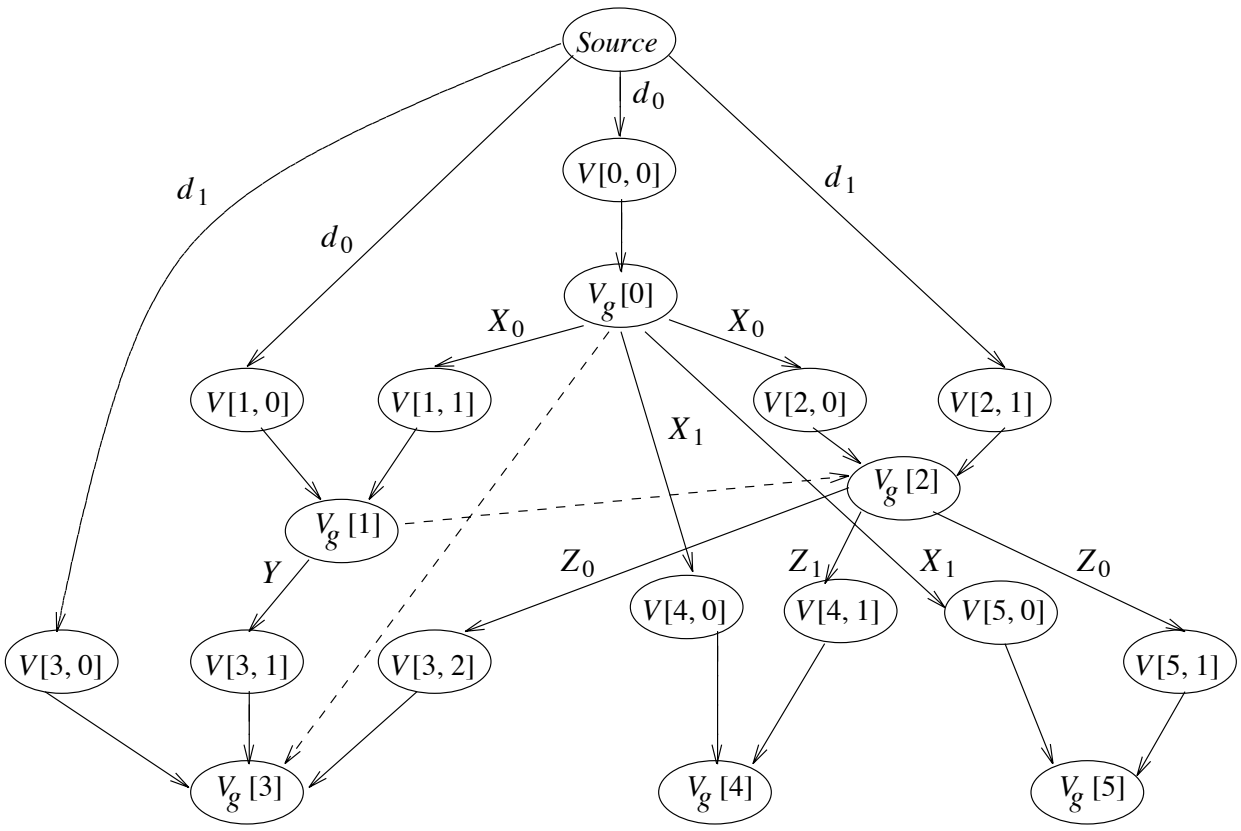

Figure 3: Generating an execution graph with respect to the given matching, scheduling, and data relocation schemes associated with the subtask ţow graph shown in Figure 1.

Step 6: Each vertex v of Ex is associated with a starting time $ST(v)$ and a Ţnishing time $FT(v)$ ($ST(v)$ and $FT(v)$ are random variables). From the deŢnitions in Steps 4 and 5, the execution graph Ex generated is acyclic. Thus, there exists a topological sort [CoL90] of the vertices in V. Set $ST(Source) = 0$. $W(v)$ is the weight of v (recall that each $V[i, j]$ is associated with a weight zero and each $V_g[i]$ is associated with a weight $C[i, Af(i)]$). Suppose that v_k is one of the immediate predecessors of v, $W(v_k, v)$ is the weight of the direct edge from v_k to v. Then $ST(v)$ and $FT(v)$ can be derived inductively one by one in the order speciŢed by the topological sort according to the following formulae:

$$ST(v) = \max_{k} \{FT(v_k) + W(v_k, v)\} \quad (1)$$

$$FT(v) = ST(v) + W(v). \quad (2)$$

Step 7: The total execution time of the application program P associated with an assignment function Af, a valid scheduling function Sf, and a valid set of data-source functions DS is deTned by the following formula:

$$\text{Execution\_time}_P(Af\ ,\ Sf\ ,\ DS\ ) = \max_{v \in V}\{FT(v)\}. \quad (3)$$

Suppose that $\underline{\text{E}\{x\}}$ denotes the expected value of a random variable x. The objective of matching, scheduling, and data relocation for HC is to Tnd an assignment function Af^*, a valid scheduling function Sf^*, and a valid set of data-source functions DS^*, such that

$\text{E}\{\text{Execution\_time}_P(Af^*, Sf^*, DS^*)\} =$

$$\min_{Af, Sf, DS} \text{E}\{\text{Execution\_time}_P(Af\ , Sf\ , DS\ )\}. \quad (4)$$

Thus, the minimization of the expected value of the total execution time of an application program is the global optimization criterion of the optimization problem for HC described in Section 1 with respect to the stochastic model deTned in Section 3.

It is assumed in this mathematical model that, if there is no data dependency between two subtasks $S[i]$ and $S[j]$, and they are assigned to be executed on two different machines by the assignment function Af, then $S[i]$ and $S[j]$ can be executed concurrently. Furthermore, the inter-machine communication step for one subtask to obtain one of its input data items can be overlapped with (a) inter-machine communication step(s) to obtain its other input data item(s), (b) the inter-machine communication steps of other subtasks to obtain their input data items, and (c) the computation steps of other subtasks. The distribution of each random variable $D[s, r, e]$ indicates any time delay resulting from network or machine I/O conticts.

As stated in Section 3, it is extremely difTcult to obtain the exact distribution of $D[s, r, e]$. The purpose of the above topological sort based procedure is not for calculating

Execution_time$_P$(*Af*, *Sf*, *DS*) in practice due to this difficulty. Rather it is to define the global optimization criterion theoretically for the optimization problem of HC. The theorem presented in Section 6 is based on this defined Execution_time$_P$(*Af*, *Sf*, *DS*) with a known *Af*, *Sf*, and *DS* and provides a practical local optimization criterion for future data relocation heuristics.

5: A Procedure for Enumerating the Valid Options in Choosing Data Relocation Schemes

5.1: Overview

In Subsection 5.2, a procedure for enumerating all the valid options in choosing the data relocation schemes with respect to an arbitrary matching is described for subtask flow graphs without inter-subtask conditional and looping constructs. With the presence of the inter-subtask conditional and looping constructs, the same procedure presented in Subsection 5.2 is extended in Subsection 5.3 to enumerate the valid options in choosing the data relocation schemes. The material presented in this section defines the search space for the optimization problem based on the stochastic model of HC mentioned in Section 1. This search space enumerates the possible combinations of *Af*, *Sf*, and *DS* with respect to a specific subtask flow graph (or a specific HC application). The number of valid combinations (i.e., the size of the search space) denotes the complexity of the optimization problem. This defined search space also helps future data relocation heuristic developers to know all the valid options in choosing a data relocation scheme.

5.2: Description for Subtask Flow Graphs without Inter-Subtask Conditional and Looping Constructs

A directed graph $\underline{Dg[Af]}$ corresponding to a specific assignment function *Af* can be generated by connecting the vertices in *V* as follows (recall that *V* is a set that contains all the vertices generated for any specific application program *P* according to Steps 1 and 2 described in Section 4). This directed graph (via vertex and edge connectivity) illustrates all possible sources from where a subtask could fetch its individual input data item:

Step 1: For every i_1, j_1, i_2, and j_2, where $0 \le i_1 < n$, $0 \le i_2 < n$, $0 \le j_1 < NI[i_1]$, $0 \le j_2 < NI[i_2]$, and $i_1 \ne i_2$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, a directed edge from $V[i_1, j_1]$ to $V[i_2, j_2]$ and a directed edge from $V[i_2, j_2]$ to $V[i_1, j_1]$ are added.

Step 2: For every i_1, j_1, i_2, and j_2, where $0 \le i_1 < n$, $0 \le i_2 < n$, $0 \le j_1 < NG[i_1]$, and $0 \le j_2 < NI[i_2]$, such that $Gd[i_1, j_1] = Id[i_2, j_2] = e$, a directed edge from $V_g[i_1]$ to $V[i_2, j_2]$ is added.

After the above Steps 1 and 2, each generated data item $Gd[i_1, j_1]$ of P corresponds to a fully connected graph of the set of vertices $\underline{VG\ [i_1,j_1]} = \{V[i_2, j_2] \mid Gd[i_1, j_1] = Id[i_2, j_2], 0 \le i_2 < n, 0 \le j_2 < NI[i_2]\}$. This corresponds to the set of input data vertices that need the generated data item $Gd[i_1, j_1]$. Also, $V_g[i_1]$ is connected uni-directionally (i.e., $V_g[i_1]$ is the starting point of each directed edge) to all the vertices in $VG[i_1, j_1]$.

Step 3: For every i, j, and k, such that $Id[i, j] = (-1, d_k)$, where $0 \le i < n$, $0 \le j < NI[i]$, and $0 \le k < Q$, a directed edge from the *Source* vertex to $V[i, j]$ is added.

After the above Step 3, each initial data item $(-1, d_k)$ $(0 \le k < Q)$ of P corresponds to a fully connected graph of the set of vertices $\underline{VI[k]} = \{V[i, j] \mid Id[i, j] = (-1, d_k)\}$ (i.e., the input data vertices that need the initial data element d_k). There is also a directed edge from the *Source* vertex to each vertex in $VI[k]$. All the edges generated in the above Steps 1, 2, and 3 are called fetch edges.

Figure 4 illustrates components of $Dg[Af]$ for the example discussed in Section 4, based on the subtask ṭow graph shown in Figure 1. Recall that the parameters of the above example subtask ṭow graph are listed by Table 1. After applying above Steps 1, 2, and 3, the edges (both solid and dashed lines) of $Dg[Af]$ in Figure 4 are fetch edges corresponding to the initial data elements d_0 and the generated data items X_0 and Z_0.

A directed graph $Dg[Af]$ can be generated by knowing only P and Af. After generating $Dg[Af]$, any algorithm for enumerating the spanning trees of a directed graph [CoL90] can be applied to the subgraphs of $Dg[Af]$ for (1) the set of vertices $\{V_g[i]\} \cup VG[i, j]$ $(0 \le i < n$ and $0 \le j < NG[i])$ and (2) the set of vertices $\{Source\} \cup VI[k]$ $(0 \le k < Q)$. The roots of all possible span-

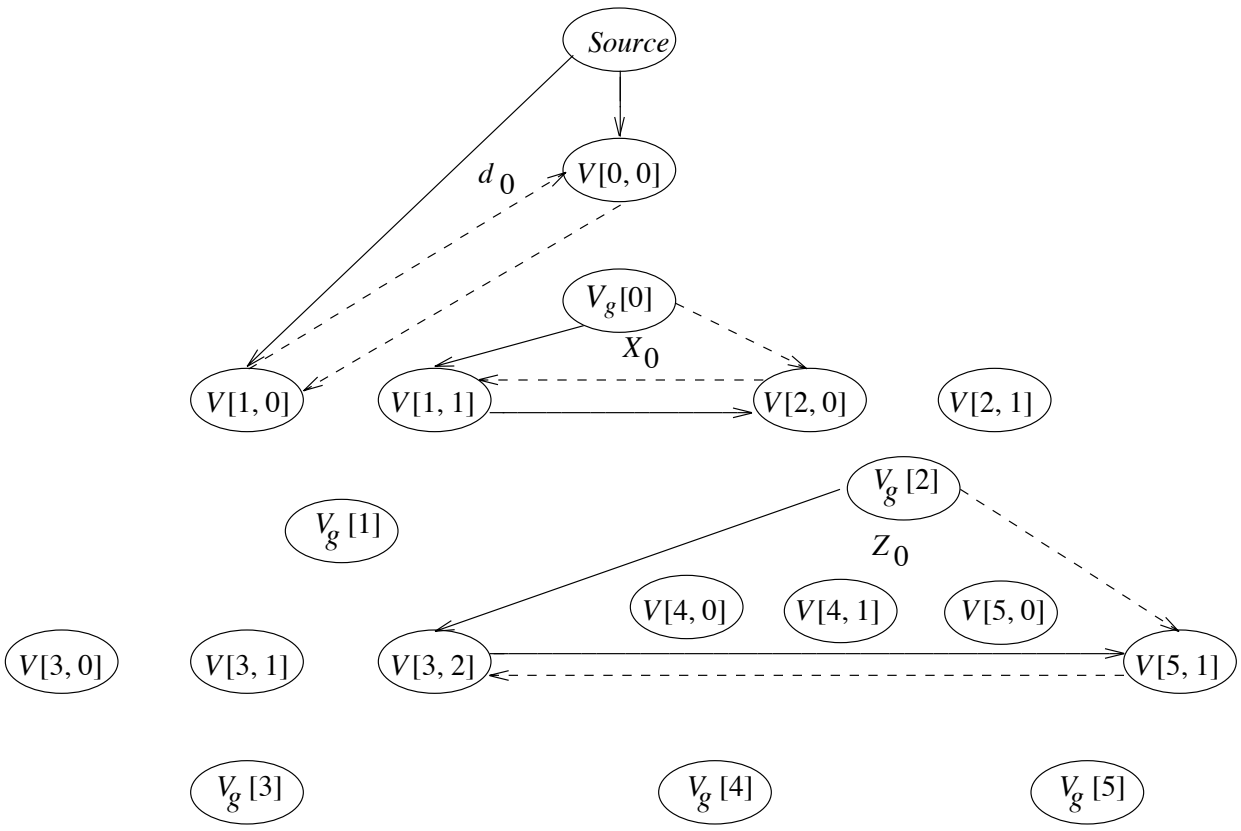

Figure 4: The d_0, X_0, and Z_0 components of $Dg[Af]$, based on the subtask ţow graph in Figure 1.

ning trees are $V_g[i]$ $(0 \le i < n)$ or the *Source* vertex, respectively. Each spanning tree corresponding to the set of vertices $\{V_g[i]\} \cup VG[i, j]$ speciŢes a valid data relocation scheme for the generated data item $Gd[i, j]$. Because the *Source* vertex can denote multiple locations where each initial data element d_k is stored before the execution of P, each spanning tree corresponding to the set of vertices $\{Source\} \cup VI[k]$ can specify a suite of valid data relocation schemes for the initial data element d_k. In the above generated spanning trees, if the parent vertex of $V[i_1, j_1]$ is $V[i_2, j_2]$ or $V_g[i_2]$, then $DS[i_1](j_1) = [i_2, Af(i_2)]$; and if the parent vertex of $V[i_1, j_1]$ is the *Source* vertex, then $DS[i_1](j_1) = [-1, q]$, where $M[q]$ is one of the initial locations of the corresponding initial data element. The solid lines in Figure 4 illustrate one spanning tree for

each of d_0, X_0, and Z_0, respectively.

5.3: Description for Subtask Flow Graphs with Inter-Subtask Conditional and Looping Constructs

5.3.1: With the Presence of Inter-Subtask Conditional Constructs

In order to maintain a static analysis approach, it is assumed that the branching probabilities $\underline{P_{\text{then}}}$ and $\underline{P_{\text{else}}}$ for the ''then'' and ''else'' clauses of the input-data-dependent conditional constructs in the subtask ţow graph are known and $P_{\text{then}} + P_{\text{else}} = 1$. Estimates of these two probabilities can be determined from empirical information or be supplied by the application users (such assumptions are typical in the literature, e.g., [Tow86]). Figure 5(a) shows an example in which there is an input-data-dependent conditional construct after $S[1]$. It is assumed that the left branch after $S[1]$ is the ''then'' clause and the right branch after $S[1]$ is the ''else'' clause of the corresponding input-data-dependent conditional construct.

The expected time for computing subtask $S[i]$ in the ''then'' clause of an input-data-dependent conditional construct on $M[Af(i)]$ is $P_{\text{then}} \cdot C[i, Af(i)]$. The expected time for computing subtask $S[i]$ in the ''else'' clause of an input-data-dependent conditional construct on $M[Af(i)]$ is $P_{\text{else}} \cdot C[i, Af(i)]$. Similarly, the inter-machine data transfer times for transferring subtasks' input and output data items inside an input-data-dependent conditional construct should be multiplied by their corresponding branching probability. For example, as shown in Figure 5(a), $(1, D_1)$ of $S[2]$ and $(4, D_5)$ of $S[5]$ are inside the input-data-dependent conditional construct, but $(0, D_0)$ of $S[1]$ and $(5, D_6)$ of $S[6]$ are not. With the above changes of the timing information, the topological sort based procedure presented in Section 4 can be used to determine the total execution time of a subtask ţow graph with input-data-dependent conditional constructs.

With the presence of input-data-dependent conditional constructs in the subtask ţow graph, the post-conditional locations of the input data items and output data items of the subtasks inside the ''then'' and ''else'' clauses cannot be determined at compile time (i.e., their locations will

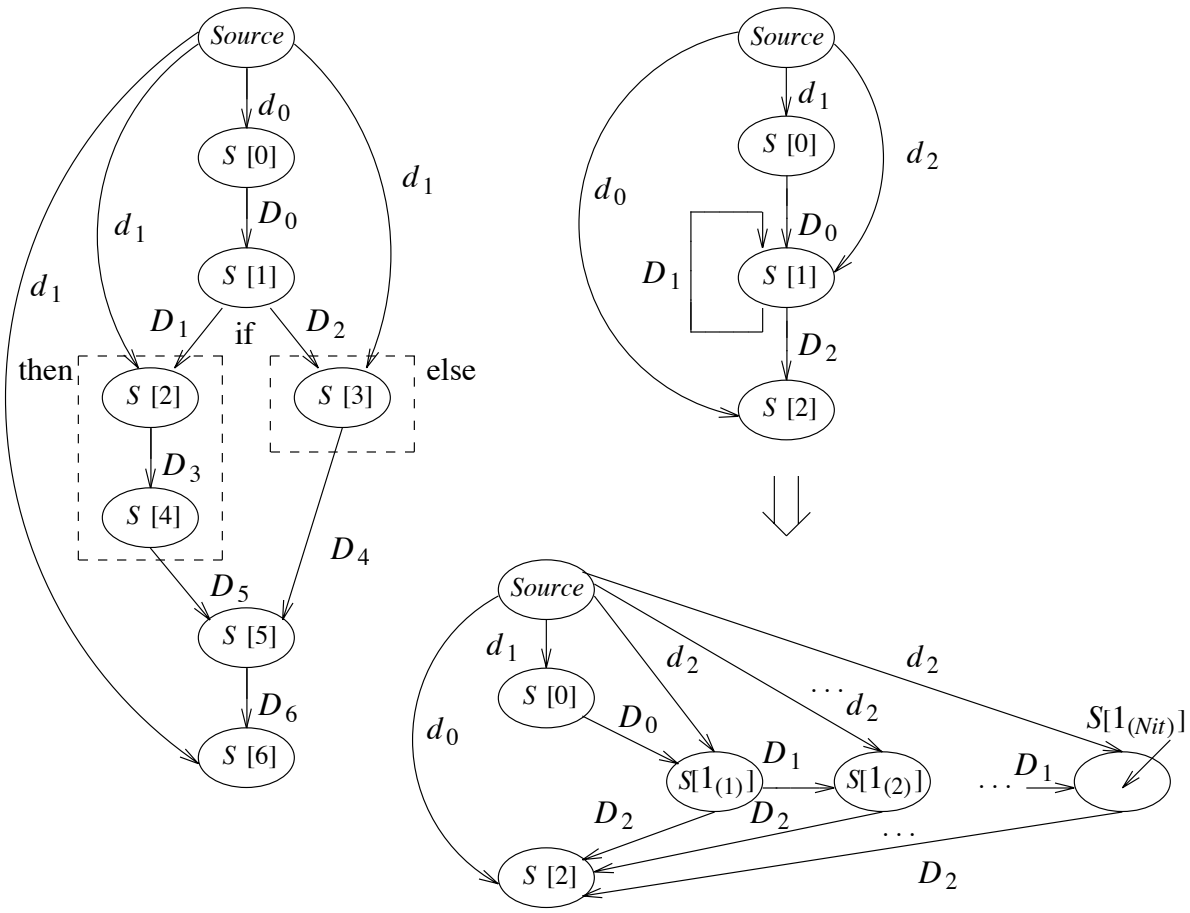

Figure 5: (a) Input-data-dependent conditional and (b) looping constructs in the subtask ṭow graphs (d_0, d_1, and d_2 are initial data elements, D_0 to D_6 are generated data items).

depend on the value of the conditional and how the clauses are executed at run time). The procedures for adding fetch edges to generate $Dg[Af]$ presented in Subsection 5.2 must be modiṬed to reṭect the properties of input-data-dependent conditional constructs. For the ease of presentation, the following procedures are presented for the case of having only one input-data-dependent conditional construct in the subtask ṭow graph. For the case of having nested input-data-dependent conditional constructs and/or more than one input-data-dependent conditional construct in the same scope level of the application program, the same procedures can be extend-

ed inductively and applied due to the modular structure [BoM76] of the subtask ţow graph.

Step 1: For each input or output data item d, an associated scope level $Scope[d]$ is deŢned. Subtasks' input and output data items that are not part of the input-data-dependent conditional construct all have their corresponding scope levels as ''Outside.'' For example, $(-1, d_0)$ of $S[0]$ and $(5, D_6)$ of $S[6]$ in Figure 5(a) belong to this category. Alternatively, subtasks' input and output data items that are part of the input-data-dependent conditional construct all have their corresponding scope levels as ''Inside.'' For example, $(1, D_1)$ of $S[2]$, $(-1, d_1)$ of $S[3]$, and $(2, D_3)$ of $S[4]$ in Figure 5(a) all belong to this category.

Step 2: Each input and output data item d is associated with a clause identiŢer $Cid[d]$. Subtasks' input and output data items that are not part of the input-data-dependent conditional construct all have their corresponding clause identiŢer as ''Global.'' For example, both $(-1, d_0)$ of $S[0]$ and $(5, D_6)$ of $S[6]$ belong to this category. Subtasks' input and output data items that are part of the ''then'' clause of the input-data-dependent conditional construct have their corresponding clause identiŢer as ''Then.'' For example, $(1, D_1)$ of $S[2]$, $(-1, d_1)$ of $S[2]$, and $(4, D_5)$ of $S[5]$ all belong to this category. Subtasks' input and output data items that are part of the ''else'' clause of the input-data-dependent conditional construct have their corresponding clause identiŢer as ''Else.'' For example, $(1, D_2)$ of $S[3]$, $(-1, d_1)$ of of $S[3]$, and $(3, D_4)$ of $S[5]$ belong to this category.

Step 3: One extension of the deŢnition of the scope level of a data item is described as follows. If for two data items $Id[i_1, j_1]$ and $Id[i_2, j_2]$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, $Scope[Id[i_1, j_1]] = Scope[Id[i_2, j_2]]$ = ''Inside,'' $Cid[Id[i_1, j_1]]$ = ''Then,'' $Cid[Id[i_2, j_2]]$ = ''Else,'' and $Af(i_1) = Af(i_2) = i$, then reset $Scope[Id[i_1, j_1]] = Scope[Id[i_2, j_2]]$ = ''Outside'' and $Cid[Id[i_1, j_1]] = Cid[Id[i_2, j_2]]$ = ''Global.'' The reason is that, because no matter what is the exact execution trace of the input-data-dependent conditional construct during run time, data item e will be available on machine $M[i]$ either via $S[i_1]$'s copy inside the ''then'' clause or $S[i_2]$'s copy inside the ''else'' clause. For example, if $Af(2) = Af(3) = i$, then $(-1, d_1)$ on machine $M[i]$ belongs to this category.

After the above three labeling steps for each data item of the subtasks in the application program, each data item d is associated with a scope level $Scope[d]$ and a clause identifier $Cid[d]$. The following Step 4 is defined based on the above two augmented parameters of d.

Step 4: Step 1 in Subsection 5.2 should be modified as the following. Steps 2 and 3 in Subsection 5.2 are unchanged.

Case A: For every i_1, j_1, i_2, and j_2, where $0 \le i_1 < n$, $0 \le i_2 < n$, $0 \le j_1 < NI[i_1]$, $0 \le j_2 < NI[i_2]$, and $i_1 \ne i_2$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, $Scope[Id[i_1, j_1]] = Scope[Id[i_2, j_2]]$, and $Cid[Id[i_1, j_1]] = Cid[Id[i_2, j_2]]$, a directed edge from $V[i_1, j_1]$ to $V[i_2, j_2]$ and a directed edge from $V[i_2, j_2]$ to $V[i_1, j_1]$ are added.

Case B: For every i_1, j_1, i_2, and j_2, where $0 \le i_1 < n$, $0 \le i_2 < n$, $0 \le j_1 < NI[i_1]$, $0 \le j_2 < NI[i_2]$, and $i_1 \ne i_2$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, $Scope[Id[i_1, j_1]] =$ ''Outside,'' and $Scope[Id[i_2, j_2]] =$ ''Inside'', only one directed edge from $V[i_1, j_1]$ to $V[i_2, j_2]$ is added.

With the above four augmenting steps (compared with Steps 1, 2, and 3 in Subsection 5.2) for generating $Dg[Af]$, subtask flow graphs with input-data-dependent conditional constructs can be handled properly. Those augmenting steps with scope levels and clause identifiers, enumerate all the possible sources for fetching each particular data item with the presence of the input-data-dependent conditional constructs in the subtask flow graph.

5.3.2: With the Presence of Inter-Subtask Looping Constructs

Similar to the case of having input-data-dependent conditional constructs, for the ease of presentation, the following procedures are presented for the case of having only one looping construct in the entire subtask flow graph. For the case of having nested looping constructs and/or more than one looping construct, the same procedures can be extended inductively and applied just as well due to the modular structure of the subtask flow graph, as discussed for the data conditional cases.

Suppose $\underline{Nit}$ is the maximum number of iterations that the looping construct will execute. $S[i_{(j)}]$ is the subtask that represents the number j iteration of subtask $S[i]$ that is inside a looping construct, for $1 \le j \le Nit$. It is assumed that the distribution for the number of iterations the looping construct will execute is known. Let $\underline{Lp[k]}$ $(0 \le k \le Nit)$ is the probability that the looping construct will execute a total of k iterations, where $\sum_{k=0}^{Nit} Lp[k] = 1$. Then, $\underline{\tilde{Lp}[j]} = \sum_{k=j}^{Nit} Lp[k]$ is the probability that iteration number j of the looping construct will be executed.

As shown in Figure 5(b), there is an input-data-dependent looping construct (containing $S[1]$) between $S[0]$ and $S[2]$. The initial cyclic subtask ţow graph (due to the presence of the looping construct) is transformed into an acyclic one. A total of Nit copies of $S[1]$ are generated.

It is assumed that a matching scheme is given for all the subtasks (including $S[i_{(j)}]$'s) in the acyclic subtask ţow graph generated by the above transformation. The time for computing iteration number j of $S[i]$ (i.e., $S[i_{(j)}]$) on machine $M[Af(i_{(j)})]$ is $\tilde{Lp}[j] \cdot C[i_{(j)}, Af(i_{(j)})]$. Similarly, the inter-machine data transfer time for transferring subtasks' input and output data items inside iteration number j of the looping construct should be multiplied by $\tilde{Lp}[j]$ as well. With the above changes of the timing information, the topological sort based procedure presented in Section 4 can be used to determine the total execution time of a subtask ţow graph with looping constructs.

For the case of having a looping construct in the subtask ţow graph, Step 1 in Subsection 5.2 must have the following two rules added. Steps 2 and 3 in Subsection 5.2 are unchanged.

(1) Subtask $S[i_{(j)}]$ (i.e., the iteration number j of $S[i]$ in the looping construct) can obtain its input data items by copying them from other subtasks that are in the iteration number k of the looping construct, where $k \le j$. $S[i_{(j)}]$ also can obtain its input data items by copying them from other subtasks that are not in the looping construct or from the *Source*.

(2) Subtask $S[k]$ that is not in the looping construct can only obtain its input data items by copying them from other subtasks that are not in the looping construct, from $S[i_{(j)}]$ where $\tilde{Lp}[j] = 1$, or from the *Source*.

6: A Greedy Approach to Establishing a Local Optimization Criterion for Developing Data Relocation Heuristics

In this section, a greedy algorithm based approach to establishing a local optimization criterion for developing data relocation heuristics is presented. This greedy strategy is established based on the mathematical model, the global optimization criterion, and search space described in Sections 3, 4, and 5, respectively, for the optimization problem in HC. The goal of this section is to show that an approach based on a greedy algorithm can establish a reasonable local optimization criterion for developing data relocation heuristics. Choosing *Af*, *Sf*, and *DS* to minimize the expected value of the total execution time based on a stochastic HC model is a complex optimization problem. However, developing heuristics to find suboptimal *Af*, *Sf*, and *DS* is necessary to use HC systems efficiently.

This section concentrates on developing data relocation heuristics to choose a suboptimal data relocation scheme for a given matching and scheduling to decrease the expected value of the total execution time of an HC application program. One of the techniques for developing heuristics is to achieve a reasonable level of global optimization using a well-evaluated local optimization criterion. The local optimization criterion for developing data relocation heuristics presented in this section is the minimization of the expected time when each subtask can start its computation after obtaining all of its input data items.

A greedy algorithm based approach for data relocation to achieving the above local optimization is to minimize the expected receiving time for each input data item of each subtask. Suppose that $\underline{rt(V[i, j])}$ is the random variable that specifies the receiving time of input data item $Id[i, j]$ for subtask $S[i]$. Then, the time when $S[i]$ can start its computation step after obtaining all of its input data items is $\max_{0 \le j < NI[i]} [rt(V[i, j])]$. The above greedy algorithm based approach states that, the minimization of $E\{\max_{0 \le j < NI[i]} [rt(V[i, j])]\}$ can be achieved by the minimization of $E\{rt(V[i, j])\}$ for all j, $0 \le j < NI[i]$. For an arbitrary set of random variables, this greedy strategy in general will *not* lead to the stated local optimization. That is, for a general set of random vari-

ables $G[j]$, $0 \le j < J$, it is *not* always the case that the minimization of $E\{\max_{0 \le j < J} G[j]\}$ can be achieved by the minimization of $E\{G[j]\}$ for all j, $0 \le j < J$. But with the assumptions of the distributions of $rt(V[i, j])$ and $rt'(V[i, j])$ corresponding to two different data relocation schemes DS and DS' shown in the next paragraph, this greedy approach can be proven to achieve the above stated local optimization.

The following assumptions about $rt(V[i, j])$ and $rt'(V[i, j])$ are made.

(1) $rt(V[i, j]) + k$ and $rt'(V[i, j]) + k'$ for a Ţxed i and j (where k and k' are arbitrary constants) belong to the same two-parameter family of random variables [CaB90]. Most of the common families of distributions for random variables, such as normal distribution, Gamma distribution, and Beta distribution, have this property.

(2) The variance of $rt(V[i, j])$ is equal to the variance of $rt'(V[i, j])$ for Ţxed i and j.

(3) For any data relocation scheme DS, $rt(V[i, j_1]) + c_1$ is independent of $rt(V[i, j_2]) + c_2$ ($j_1 \neq j_2$ and c_1 and c_2 are arbitrary constants).

Assumptions (1), (2), and (3) are all related to the statistical properties of $rt(V[i, j])$ and $rt'(V[i, j])$. If these assumptions are approximately satisŢed in reality, the theorem that follows based on these assumptions is of practical as well as theoretical signiŢcance. For assumptions (1) and (2), because $rt(V[i, j])$ and $rt'(V[i, j])$ are two random variables for specifying the receiving times of the *same* data item (i.e., $Id[i, j]$) for $S[i]$, for the same Af and Sf, but corresponding to two different data relocation schemes, it is quite reasonable to assume that they have certain similar statistical properties (e.g., their variances, their families of distribution). For assumption (3), although $rt(V[i, j_1])$ and $rt(V[i, j_2])$ are deŢned for two different data items, if the inter-machine data transfer steps for $Id[i, j_1]$ and $Id[i, j_2]$ will impact each other or those two data items are generated by the same subtask, their corresponding receiving times by $S[i]$ can be correlated to each other. However, conditions exist under which the random variables can be treated as being independent of each other despite this type of correlation. The Kleinrock in-

dependence approximation for a data network in which there are many interacting transmission queues [Kle64] is a well-known method for describing this situation. This Kleinrock independence approximation is used here as the basis for assuming independence between $rt(V[i, j]) + c_1$ and $rt'(V[i, j]) + c_2$ that may technically be correlated. In [LiA97], similar assumptions are made about the execution time distributions for the individual subtasks for statically estimating the execution time distribution for an entire HC application program.

Theorem: For two different data relocation schemes *DS* and *DS'*, with the same *Af* and *Sf*, and a fixed i ($0 \le i < n$), suppose $\underline{X_j} = rt(V[i, j])$, $\underline{Y_j} = rt'(V[i, j])$, $\underline{X} = \max_j[X_j]$, and $\underline{Y} = \max_j[Y_j]$ ($0 \le j < NI[i]$), where X and Y are random variables for specifying the times when $S[i]$ receives all of its input data items with respect to *DS* and *DS'*. If $E\{X_j\} \le E\{Y_j\}$ for $0 \le j < NI[i]$, then $E\{X\} \le E\{Y\}$.

Proof: Suppose that the distribution function of a random variable w is F_w. Because $E\{X_j\} \le E\{Y_j\}$ for all j, there exists $c_j \ge 0$, such that $E\{X_j\} + c_j = E\{X_j + c_j\} = E\{Y_j\}$. Due to assumption (2), $Var\{X_j + c_j\} = Var\{X_j\} = Var\{Y_j\}$. Then, because of assumption (1), $F_{X_j + c_j} = F_{Y_j}$. From assumption (3), for $0 \le j < NI[i]$, $\{X_j + c_j\}$ and $\{Y_j\}$ are two sets of independent random variables. With the properties associated with the ''max'' operator over multiple independent random variables [CaB90], it can be shown that

$$F_{\max\{X_j + c_j\}} = \prod_{j=0}^{NI[i]-1} F_{X_j + c_j} = \prod_{j=0}^{NI[i]-1} F_{Y_j} = F_{\max\{Y_j\}}.$$

Therefore, $E\{Y\} = E\{\max_j[Y_j]\} = E\{\max_j[X_j + c_j]\}$. Because $c_j \ge 0$, $E\{Y\} = E\{\max_j[X_j + c_j]\} \ge E\{\max_j[X_j]\} = E\{X\}$. Thus, $E\{X\} \le E\{Y\}$. □

Based on the above theorem, the greedy algorithm based approach that finds a data relocation scheme to minimize $E\{rt(V[i, j])\}$ for $S[i]$ to obtain $Id[i, j]$ with respect to the same *Af* and *Sf* for all j ($0 \le j < NI[i]$) can also minimize the expected time when $S[i]$ receives all of its input data items (i.e., $E\{X\}$) and is ready for its computation. The exact starting time and the cost of the computation for $S[i]$ (i.e., $ST(V_g[i])$ and $C[i, Af(i)]$) depend on the choice of *Af* and *Sf*.

The signiȚcance of the above theorem is that it shows a greedy algorithm based approach can establish a reasonable local optimization criterion for developing data relocation heuristics. Based on the above conclusion, in order to minimize the expected total execution time of an application program executed in a dedicated HC system (the global optimization criterion), data relocation heuristics should select the source for each input data item of $S[i]$, among all the valid options described in Section 5, such that its receiving time by $S[i]$ is as small as possible. With this greedy approach, the expected time when $S[i]$ can start its computation after obtaining all of its input data items (the local optimization criterion) can be minimized.

Given the approximation assumptions made, theoretically there exists a *DS* that can satisfy $\mathrm{E}\{X_j\} \le \mathrm{E}\{Y_j\}$ for $0 \le j < NI[i]$. However, in a real HC system, the inter-machine communication steps speciȚed by the selected data relocation scheme for one subtask may impact the expected receiving time of input data items for other subtasks. Thus, the data relocation scheme that minimizes $E\{rt(V[i, j])\}$ for every $S[i]$ and all j ($0 \le j < NI[i]$) may be hard to Țnd or may not exist in practice. Trade-offs must be made to choose a suboptimal data relocation scheme, such that more input data items can be obtained by more subtasks as quickly as possible.

Because the $rt(V[i, j])$'s corresponding to different data relocation schemes are random variables, in general, the distributions of the $rt(V[i, j])$'s are needed for choosing a proper *DS* for obtaining $Id[i, j]$ for $S[i]$. But based on the above proven theorem with its underlying assumptions, only the expected values of $rt(V[i, j])$'s are needed to make such a decision, because the minimization of the expected time when $S[i]$ can obtain $Id[i, j]$ for each $0 \le j < NI[i]$ can lead to the minimization of the expected time when $S[i]$ obtains all of its input data items and may start its computation. In practice, $E\{rt(V[i, j])\}$ corresponding to a particular *DS* is much easier to estimate than the probability distribution function of $rt(V[i, j])$. For example, for the networks using the popular and ubiquitous TCP/IP protocol, the inter-machine communication time component of $E\{rt(V[i, j])\}$ can be estimated by the value of the expected round trip time (RTT) delay between any two machines in the network kept by the TCP protocol on routers [Com91].

The minimization of the expected time when a selected subtask $S[i]$ can start its computation is adopted by many other matching and scheduling heuristics as their local optimization criterion [ElL90, IvO95, SiL93, WaA96]. The greedy approach, validated by the above theorem, for achieving this local optimization can be used by those matching and scheduling heuristics to intelligently select a data relocation scheme for $S[i]$ based on a theoretical stochastic HC model. Thus, coupled with the selection criterion for specifying the order of achieving the above local optimization for subtasks in the original matching and scheduling heuristic (e.g., priority-based for list scheduling), the greedy strategy for developing a data relocation heuristic presented in this paper can be expected to further decrease the inter-machine communication overhead of the given HC application program. For the matching and scheduling related heuristics that do not adopt the above local optimization criterion to achieve global optimization, choosing a data relocation scheme to minimize the expected time when a selected $S[i]$ can start its computation (realized by the presented greedy strategy) is still a reasonable approach to decrease the inter-machine communication overhead of an HC application. For example, in [WaS96], a genetic-algorithm-based heuristic for matching and scheduling applies the greedy strategy presented in this paper for selecting data relocation for a given matching and scheduling.

7: Summary

In an HC system, the subtasks of an application program P must be assigned to a suite of heterogeneous machines (the matching problem) and ordered (the scheduling problem) to utilize computational resources effectively. The matching and scheduling solutions presented in the literature, in general, concentrate on decreasing the computation time of P. The inter-machine communication time of P is impacted by the scheme for distributing the initial data elements and the generated data items of P to different subtasks (the data relocation problem).

The inter-machine communication time in an HC system can have a signiȚcant impact on overall system performance, so that any technique that can be used to reduce this time is important. This paper focused on the data relocation scheme to decrease the inter-machine communi-

cation time for given matching and scheduling schemes, when the possible concurrent execution of multiple subtasks on different machines is considered.

This paper concentrates on theoretical aspects of data relocation using a stochastic HC model. The optimization problem for minimizing the total execution time of an application program executed in a dedicated HC system with respect to matching, scheduling, and data relocation is completely deTned. This theoretical deTnition is based on the stochastic mathematical model, the global optimization criterion, and the search space described in Sections 3, 4, and 5, respectively. The cases in which the application programs may include inter-subtask conditional and looping constructs are considered. The practical application of the above theoretical results is demonstrated by the theorem shown in Section 6 that validates a greedy algorithm based approach can establish a reasonable local optimization criterion for developing data relocation heuristics. The validation is provided by a theoretical proof based on a set of common assumptions about the underlying HC system and application program. The stochastic HC model presented, the local optimization criterion established by the greedy approach, and the search space deTned for choosing valid data relocation schemes can help developers of future data relocation heuristics.

Acknowledgments: The authors thank John K. Antonio, Gayathri Krishnamurthy, Mark B. Kulaczewski, Yan A. Li, and Janet M. Siegel for their valuable comments. A preliminary version of portions of this material was presented at the 6th Heterogeneous Computing Workshop. Shoukat Ali assisted in the preparation of the Tnal manuscript.

Appendix: Glossary of Notation

Af	assignment function (assigns subtasks of application program P to machines)
$C[i, j]$	computation time of subtask i on machine j
$Cid[d]$	clause identiTer for data item d
$Dg[Af]$	directed graph (corresponding to program P) showing data transfer options based on a given Af
d_k	k-th initial data element of the application program P
$DS[i](j)$	source of the j-th input data item for subtask i
$D[s, r, e]$	time for transferring data item e from machine s to machine r
Ex	generated execution graph corresponding to a particular Af, Sf, and DS
$FT(v)$	Tnishing time of data transfer step (associated with an input data vertex v) and computation step (associated with an output data vertex v)
$G[i]$	generated output data set of subtask i
$Gd[i, j]$	j-th generated output data item of subtask i
Gr	generated graph corresponding to a particular Af and DS
$I[i]$	input data set of subtask i
$Id[i, j]$	j-th input data item of subtask i
$Lp[k]$	probability that a looping construct will execute a total of k iterations
$\tilde{Lp}[j]$	probability that iteration number j of a looping construct will be executed
$M[j]$	j-th machine in the HC system, $0 \leq j < m$
m	number of machines in the HC system
n	number of subtasks in the application program P

$NG[i]$	number of output data items generated by subtask i
Nit	maximum number of iterations that a looping construct will execute
$NI[i]$	number of input data items required by subtask i
$NS[j]$	number of subtasks assigned to be executed on machine j
P_{then}	branching probability for the ''then'' clause of the inter-subtask input-data-dependent conditional construct
P_{else}	branching probability for the ''else'' clause of the inter-subtask input-data-dependent conditional construct
Q	number of initial data elements for the application program P
$rt(V[i, j])$	random variable that specifies the receiving time of input data item $Id[i, j]$ for subtask $S[i]$
$Scope[d]$	scope level of data item d
Sf	scheduling function associated with the application program P
$S[i]$	i-th subtask of an application program P, $0 \leq i < n$
$S[i_{(j)}]$	subtask that represents the number j iteration of subtask $S[i]$ that is inside a looping construct
$ST(v)$	starting time of data transfer step (associated with an input data vertex v) and computation step (associated with an output data vertex v)
$V_g[i]$	output data vertex of subtask i
$V[i, j]$	j-th input data vertex of subtask i
$VG[i, j]$	set of input data vertices that need the generated data item $Gd[i, j]$
$VI[k]$	set of input data vertices that need the initial data element d_k
$W(v)$	weight of an input or output data vertex v
$W(v_k, v)$	weight of the direct edge from v_k to v, where v_k is one of the immediate predecessors of v

References:

[BoM76] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, Elsevier Science Publishing Co., Inc., New York, NY, 1976.

[CaB90] G. Casella and R. L. Berger, *Statistical Inference*, Wadsworth & Brooks/Cole Advanced Books & Software, PaciTc Grove, CA, 1990.

[ChE93] S. Chen, M. M. Eshaghian, A. Khokhar, and M. E. Shaaban, ''A selection theory and methodology for heterogeneous supercomputing,'' *2nd Workshop on Heterogeneous Processing*, Apr. 1993, pp. 15-22.

[CiL95] M. Cierniak, W. Li, and M. J. Zaki, ''Loop scheduling for heterogeneity,'' *4th IEEE Int'l Symp. on High-Performance Distributed Computing*, Aug. 1995, pp. 78-85.

[CoL90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[Com91] D. E. Comer, *Internetworking with TCP/IP, Volume I: Principles, Protocols, and Architecture*, Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1991.

[ElL90] H. El-Rewini and T. G. Lewis, ''Scheduling parallel program tasks onto arbitrary target machines,'' *J. of Parallel and Distributed Computing*, Vol. 9, No. 2, June 1990, pp. 138-153.

[Fre89] R. F. Freund, ''Optimal selection theory for superconcurrency,'' *Supercomputing '89*, Nov. 1989, pp. 699-703.

[FrG98] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, ''Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet,'' *7th Heterogeneous Computing Workshop,* Mar. 1998, pp. 184-199.

[FrS93] R. F. Freund and H. J. Siegel, ''Heterogeneous processing,'' *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13-17.

[GhY93] A. Ghafoor and J. Yang, ‘‘Distributed heterogeneous supercomputing management system,’’ *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78-86.

[IvO95] M. A. Iverson, F. Ozguner, and G. J. Follen, ‘‘Parallelizing existing applications in a distributed heterogeneous environment,’’ *4th Heterogeneous Computing Workshop*, Apr. 1995, pp. 93-100.

[KhP92] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, ‘‘Heterogeneous supercomputing: problems and issues,’’ *1st Workshop on Heterogeneous Processing*, Mar. 1992, pp. 3-12.

[KhP93] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, ‘‘Heterogeneous computing: challenges and opportunities,’’ *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

[Kle64] L. Kleinrock, *Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, New York, NY, 1964.

[LiA97a] Y. A. Li and J. K. Antonio, ‘‘Estimating the execution time distribution for a task graph in a heterogeneous computing system,’’ *6th Heterogeneous Computing Workshop*, Apr. 1997, pp. 172-184.

[LiA97b] Y. A. Li, J. K. Antonio, H. J. Siegel, M. Tan, and D. W. Watson, ‘‘Determining the execution time distribution for a data parallel program in a heterogeneous computing environment,’’ *J. of Parallel and Distributed Computing*, Vol. 44, No. 1, July 1997, pp. 35-52.

[NaY94] B. Narahari, A. Youssef, and H. A. Choi, ‘‘Matching and scheduling in a generalized optimal selection theory,’’ *3rd Heterogeneous Computing Workshop*, Apr. 1994, pp. 3-8.

[SiA96] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, ‘‘Heterogeneous computing,’’ in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

[SiD97] H. J. Siegel, H. G. Dietz, and J. K. Antonio, ‘‘Software Support for Heterogeneous Computing,’’ in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

[SiL93] G. C. Sih and E. A. Lee, ‘‘A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures,’’ *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 2, Feb. 1993, pp. 75-87.

[TaS97] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, ‘‘Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system,’’ *IEEE Trans. on Parallel and Distributed Systems*, Vol. 8, No. 8 Aug. 1997, pp. 857-871.

[Tow86] D. Towsley, ‘‘Allocating programs containing branches and loops within a multiple processor system,’’ *IEEE Trans. on Software Engineering*, Vol. SE-12, No. 10, Oct. 1986, pp. 1018-1024.

[WaA96] D. W. Watson, J. K. Antonio, H. J. Siegel, R. Gupta, and M. J. Atallah, ‘‘Static matching of ordered program segments to dedicated machines in a heterogeneous computing environment,’’ *5th Heterogeneous Computing Workshop*, Apr. 1996, pp. 24-37.

[WaK92] M. Wang, S.-D. Kim, M. A. Nichols, R. F. Freund, H. J. Siegel, and W. G. Nation, ‘‘Augmenting the optimal selection theory for superconcurrency,’’ *1st Workshop on Heterogeneous Processing*, Mar. 1992, pp. 13-22.

[WaS97] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, ‘‘Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach,’’ *J. of Parallel and Distributed Computing,* Special Issue on Parallel Evolutionary Computing, Vol. 47, No. 1, Nov. 25, 1997, pp. 8-22.

[YaK94] J. Yang, A. Khokhar, S. Sheikh, and A. Ghafoor, ‘‘Estimating execution time for parallel tasks in heterogeneous processing (HP) environment,’’ *3rd Heterogeneous*

Computing Workshop, Apr. 1994, pp. 23-28.

[ZhY95] X. Zhang and Y. Yan, ‘‘Modeling and characterizing parallel computing performance on heterogeneous networks of workstations,’’ *7th IEEE Symp. on Parallel and Distributed Processing*, Oct. 1995, pp. 25-34.